CSC263H	*Data Structures and Analysis*	Note on Bounds
University of Toronto		

Worst-case Time Complexity: Proving Asymptotic Bounds.

Let $t(x)$ be the number of steps taken by algorithm $\mathcal{A}$ on input x.
Let $T(n)$ be the *worst-case* time complexity of algorithm $\mathcal{A}$:

$$T(n) = \max_{\text{all inputs } x \text{ of size } n} t(x) \;=\; \max\{t(x) : x \text{ is an input of size } n\}$$

1. To prove that $T(n)$ is $O(g(n))$, one must show that there is a constant $c > 0$, and an input size $n_0 > 0$, such that for all $n \geq n_0$:

$$\begin{array}{ll} & T(n) \leq c \cdot g(n) \\ \Leftrightarrow & \max\{t(x) : x \text{ is an input of size } n\} \leq c \cdot g(n) \\ \Leftrightarrow & \text{For } \textit{every} \text{ input } x \text{ of size } n,\ t(x) \leq c \cdot g(n) \\ \Leftrightarrow & \text{For } \textit{every} \text{ input of size } n,\ \mathcal{A} \text{ takes } \textit{at most } c \cdot g(n) \text{ steps} \end{array}$$

2. To prove that $T(n)$ is $\Omega(g(n))$, one must show that there is a constant $c > 0$, and an input size $n_0 > 0$, such that for all $n \geq n_0$:

$$\begin{array}{ll} & T(n) \geq c \cdot g(n) \\ \Leftrightarrow & \max\{t(x) : x \text{ is an input of size } n\} \geq c \cdot g(n) \\ \Leftrightarrow & \text{For } \textit{some} \text{ input } x \text{ of size } n,\ t(x) \geq c \cdot g(n) \\ \Leftrightarrow & \text{For } \textit{some} \text{ input of size } n,\ \mathcal{A} \text{ takes } \textit{at least } c \cdot g(n) \text{ steps} \end{array}$$

IN SUMMARY:

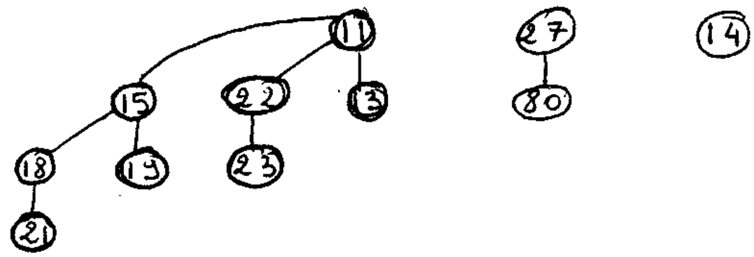
Let $T(n)$ be the *worst-case* time complexity of algorithm $\mathcal{A}$.

order

1. $T(n)$ is $O(g(n))$ iff $\exists c > 0$, $\exists n_0 > 0$, such that $\forall n \geq n_0$:
   for *every* input of size n, $\mathcal{A}$ takes *at most* $c \cdot g(n)$ steps.
2. $T(n)$ is $\Omega(g(n))$ iff $\exists c > 0$, $\exists n_0 > 0$, such that $\forall n \geq n_0$:
   for *some* input of size n, $\mathcal{A}$ takes *at least* $c \cdot g(n)$ steps.
3. $T(n)$ is $\Theta(g(n))$ iff $T(n)$ is $O(g(n))$ and $T(n)$ is $\Omega(g(n))$.

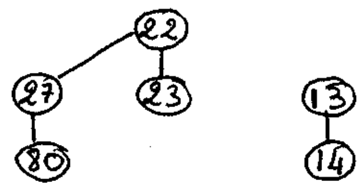
DELETE_MIN(T)

Example : T =



Delete_MIN(T) :
1. Scan roots to find smallest element
2. Delete element
3. Merge resulting BQ's.

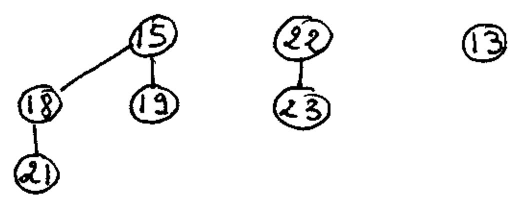
Carry :



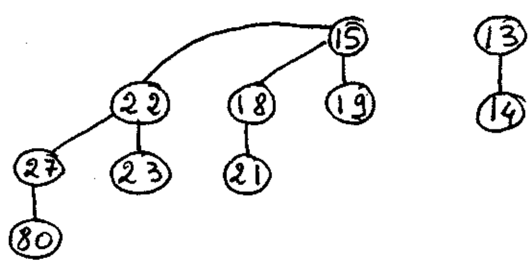
$T_1 = T - S_3$

+

$T_2 = S_3 - \{x\}$

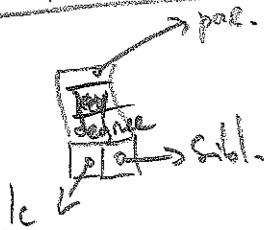


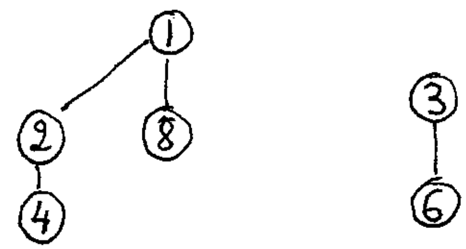
$T \leftarrow \text{UNION}(T_1, T_2)$



EXAMPLE OF BINOMIAL TREE MERGE

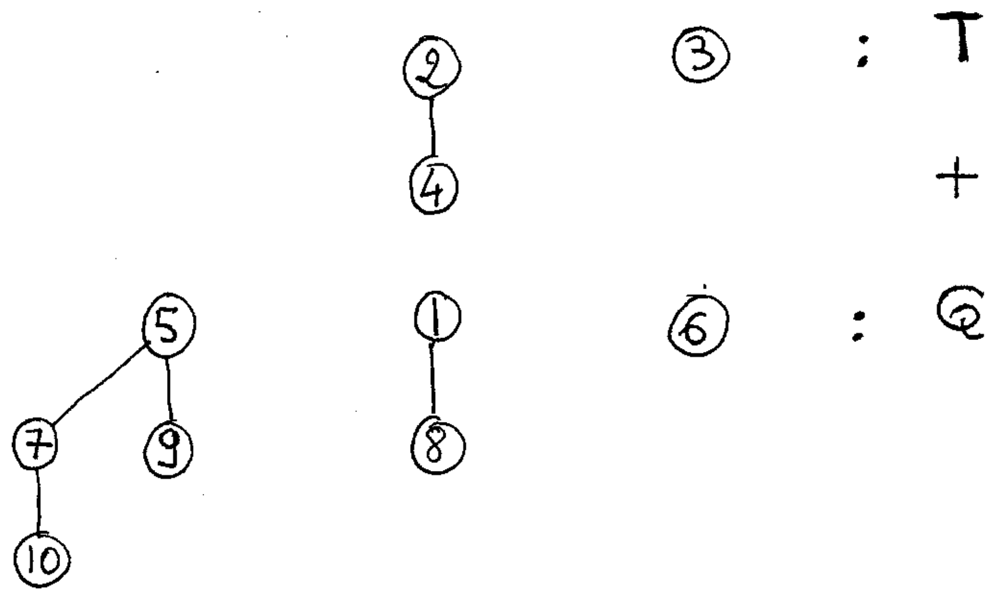
Implementation



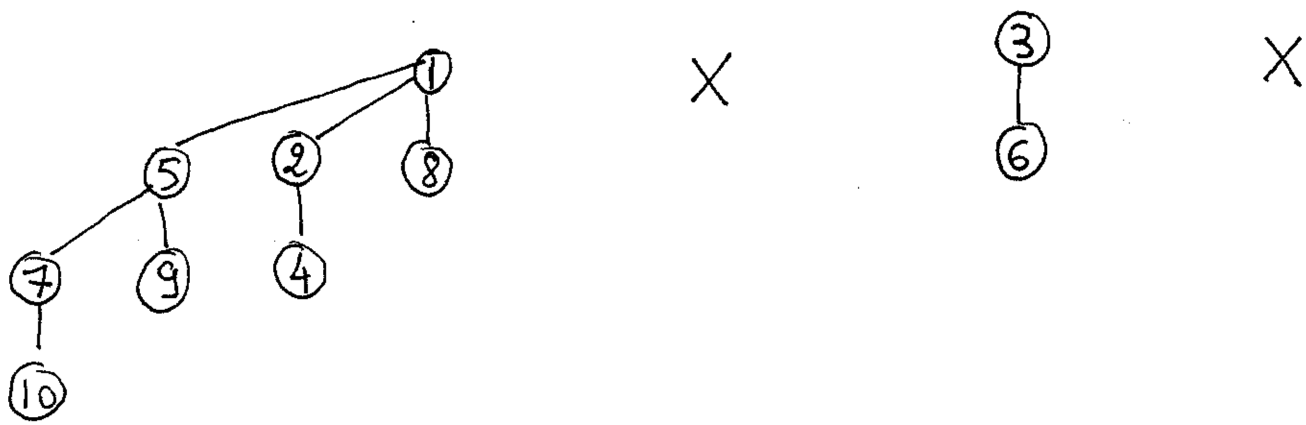


Carry

Total of 5 edges



Total of 8 edges



Merge took 8 - 5 = 3 key comparisons

ADT	Descrip	DT
priority Queue	◦ Set S with elem't/keys ◦ insert (S, x) ◦ max (s) ◦ extract max (s)	◦ unordered ll → INS $\in \Theta(1)$ → ex-max $\in \Theta(n)$ ◦ ordered ll → ins $\in \Theta(n)$ → ex-max $\in \Theta(1)$ ◦ heap (max, min) → INS, exmax $\in \Theta(\lg n)$
Mergeable P Q	u ◦ Union (S, T)	◦ binomial heap → ins $\in \Theta(\lg n)$ → ex. min $\in \Theta(\lg n)$ → union $\in \Theta(\lg n)$
Dictionary	◦ Set ◦	◦ BST → ? $O(n)$ ◦ balanced BST ↳ AVL-tree ◦

DT: binary $\overset{max}{\vee}$ heap : CBT ( ) + max heap property

~~$C[\text{heap}] = C[\text{min-heap}] + C[\text{max-heap}]$~~

BH $\subset$ H, BH = min BH $\cup$ max BH, H = min H $\cup$ max H

binomial tree $S_k = \begin{cases} o & \text{if } k=0 \\ \Delta_{S_{k-1}} - \Delta_{S_{k-1}} & \text{if } k>0 \end{cases}$

(2^k nodes, $\binom{k}{d}$ at depth d.

binomial forest $BF_m = (S_k \,\|\, |\cup S_k| = m)$

BT →

BST → | total order over in order

CBT →

FBT →

$\underbrace{\text{AA, AVL, red-black, 2-3-4}}_{\text{self balancing}}$

sb-bst

Data structures

$\boxed{\text{ADT}}$: descrip° of object and ops

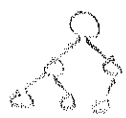
$\boxed{\text{DS}}$: specific implementa° of ADT

e.g. Priority Queues → obj: set S of elem^t w/ keys
→ op's: insert(S, x) : $(S \leftarrow S \cup \{x\})$
max(S) : $x \leftarrow \max(S)$
extract-max(S) : $\leftarrow \max(S)$
$(S \leftarrow S - \{x\})$

→ DT for PQ

e.g.	ins(S, x)	extract-max(S)
unordered linked list	$\Theta(1)$	$\Theta(n)$
ordered "	$\Theta(n)$	$\Theta(1)$
heap	$\Theta(\log n)$	$\Theta(\log n)$

$\boxed{\text{CBT}}$:



$\boxed{\text{Max-Heap}}$: CBT, s.t. $\forall n \in CBT$, value(n) $\geq$ value(Lchild(n))
$\geq$ value(Rchild(n))

max Heap property

→ DDT:

1	2	3	4	5	6	

heapsize == 9

Lchild of A[i] is A[2i]
Rchild " A[2i+1]

parent: $A[\lfloor \frac{i}{2} \rfloor]$

height of ~~tree~~ is length of longest path from root to leaf.

→ Heap Ops constraints: 1. Get CBT shape right
2. Get maxheapp. right

Insert: put at last pos then swap (max: height)

extract-max: swap last pos with root
: decrease heapsize
: heapify (swap, compare w/ heapsize)

ADT	Ops	DS
priority Q	· insert · max · extract_max	Heap (CLRS6)
mergeable priority Q	All above + Union(S, T)	Binomial Heaps

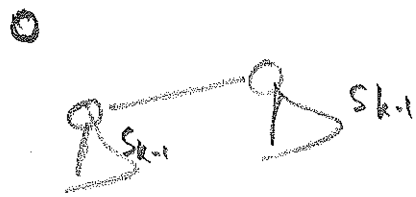
of — based on CBT bin forest

CSC 263 COG 250
CSC 321 MAT 224
CSC 369 PSY 270
CSC 2534 ECE 516

S_k - Trees

$S_0 \Rightarrow$ o

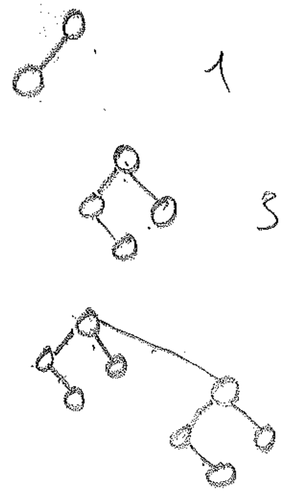
$S_k \Rightarrow$



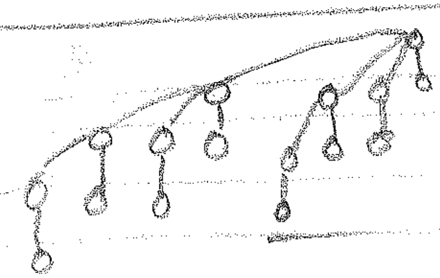
e.g. $S_1 \rightarrow$ 1

$S_2 \rightarrow$ 3

$S_3 \rightarrow$ 7



S_4:



$\binom{4}{0} = 1$
$\binom{4}{1} = 4$
$\binom{4}{2} = 6$
$\binom{4}{3} = 4$
$\binom{4}{4} = 1$

bin coefs

S_k-tree has 2^k nodes and $\binom{k}{d}$ nodes at depth d.

Binomial Forest

A binomial forest of size n (BF_n) is a sequence of S_k-trees w/ decreasing k's and s nodes

e.g. ① $BF_7 = \langle S_2, S_1, S_0 \rangle$
② $BF_9 = \langle S_3, S_0 \rangle$

S_k tree has a root node w/ exactly k children

A BF_n with $n = \langle b_\ell, \ldots, b_0 \rangle_2$ nodes
$BF_n = \langle$ all S_k-trees s.t. $b_k = 1 \rangle$ ⟶ ?

① Largest tree in BF_n is S_ℓ with $\ell = \lfloor \log_2 n \rfloor$
② Let $\alpha(n) = \#$ of 1's in the binary rep of n
  ⓐ BF_n has $\alpha(n)$ trees
  ⓑ BF_n has $n - \alpha(n)$ edges

A (min) binomial heap of n elements with keys is a BF_n s.t.
  ⓐ Each node of BF_n stores 1 element
  ⓑ Each S_k-tree of BF_n is (min) heap-ordered

log n
time

(1) $S \leftarrow$ UNION (S, T) → binary additio

(2) insert → form $\{x\}$ → S_0: (x) → Union

(3) MIN(T) → look at roots $\left(O(\log n) \right)$

(4) Extract_min / Delete_min

e.g. $|T| = 27$ → $\langle S_4, S_3, S_1, S_0 \rangle$ (11011)

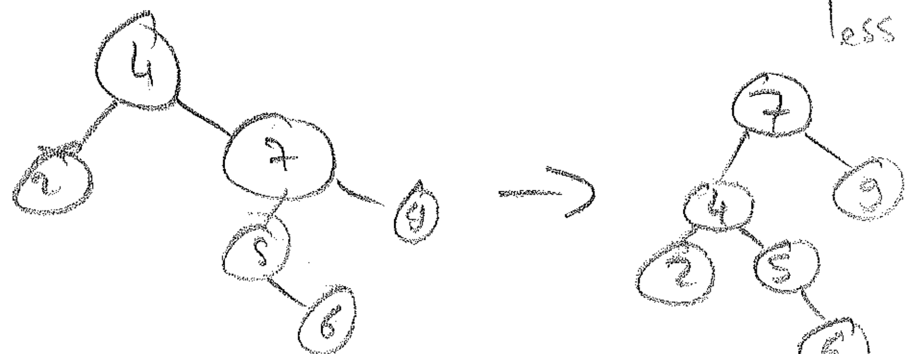
→ $27 - 4 = 23$ edges

insert $\{x\}$: $\begin{array}{r} 11011 \\ + \quad\quad 1 \\ \hline 11100 \end{array}$ } 2 key comps ⇒ 2 new edges

⇒ 25 edges (23+2) (check: $28 - 3 = 25$)

→ Bin heap: since based on bin additio
→ when insertions > log n ⇒ avg insert is cst
less than 2

BST rotatio



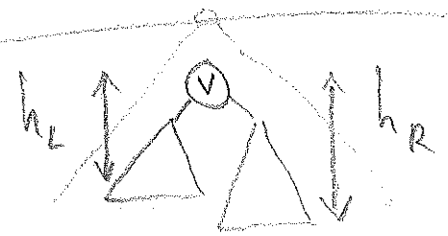
Priority Queues implemented with heaps (and used almost always for that purpose).

$\log_2 i$

e^{-2}

ADT	OPS	Data Structures
Dictionary	Insert, Delete, Search	BST
		Balanced BST · 2-3 trees · Red-black trees · AVL-trees (...)

→ Balanced trees



$BF(v) = h_R - h_L$

height Right subtree − height left subtree

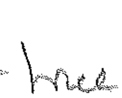
AVL-tree == BST s.t. $\forall v \in AVL$, $BF(v) = -1 \,||\, 0 \,||\, +1$

height of AVL : with n nodes have height $\Theta(\log n)$

$[\leq 1.44 \log(n+2)]$

→ Can do inserts and deletes while Maintaining the balance

insert(T, x)
- insert as in any BST, x == leaf
- go up to root # Notice: if encounter 0, no need to go up any more #
- → rotate if needed

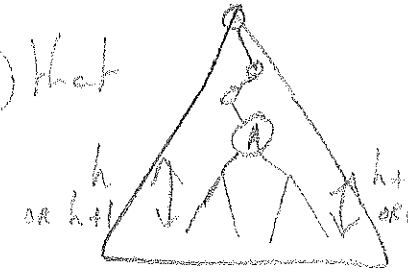
→ 

After Rotation:
(1) Rebalanced
(2) BST property
(3) height same as before (bonus)

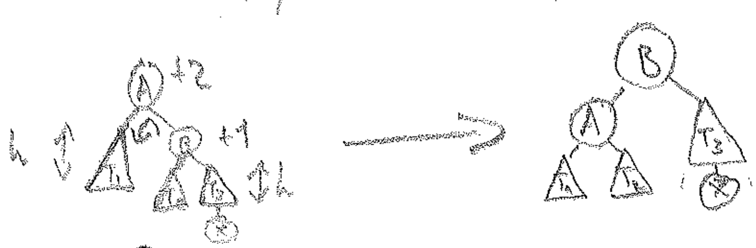
1086pm

Rebalancing

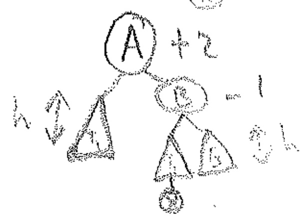
Let (A) be the first node on path (x) ⇝ root(T) that becomes unbalanced.



- wlog (A) +1 ⇝ +2 (sym. (A) −1 ⇝ −2)
  - case (1).1



  - case (1).2



Deletion

No bonus

Article on Sorting + DS

for the CSC 150

236SLOG → 263SLOG

TSP

ב/ב כ/ך מ/ם פ/ף צ/ץ ת/ת

aleph bet gimel dalet he vav zayin het tet yod kaf lamed mem nun samekh ayin pe sadhe qof resh shin taw

א ב ג ד ה ו ז ח ט י כ ל מ נ ס ע פ צ ק ר ש ת

Α Β Γ Δ Ε Ζ Η Θ Ι Κ Λ Μ Ν Ξ Ο Π Ρ Σ Τ Υ Φ Χ Ψ Ω

α β γ δ ε ζ η θ ι κ λ μ ν ξ ο π ρ σ τ υ φ χ ψ ω

5

Summary

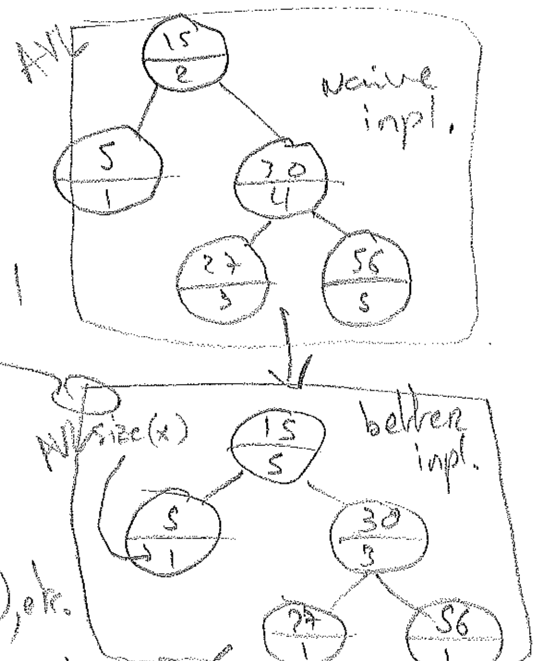
1	Priority Qs	Heaps
2	Mergeable priority Qs	Binomial Heaps
3	Dictionary	BSTs, AVL

Augmenting Data Structures

E.g. Dynamic Order Statistics

- set S of n elements with keys
- ops: insert, delete, search
  - select(k): return the element of rank k. (k^{th} one in sorted order)
  - rank(k): returns the rank of k.

e.g. S = {5, 15, 27, 30, 56}
select(4) = 30
rank(15) = 2



size(x) = size(left(x)) + S(R(x)) + 1

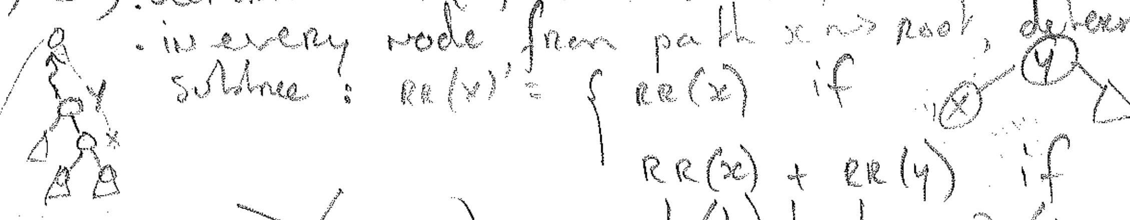
relative ranking

RR(x) = size(left(x)) + 1

select(k) ⟹ to find k^{th} rank,
if k < RR(x), find k^{th} rank of left(x), etc.

⟹ (since AVL tree), select(k) takes O(log n) in WC.

rank(k) ⟹ . determine RR(x) in x-subtree
. in every node from path x ⟶ root, determine RR(x) in y-subtree: RR(x)' = { RR(x) if ; RR(x) + RR(y) if



⟹ (——), rank(k) takes O(log n) in WC.

Maintaining

Maintaining size info 

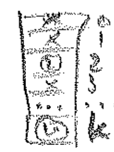
- insert(x):
  - x ends up a leaf $\Rightarrow$ size(x) := 1
  - add 1 to size(y) for all $y \in$ path $x \rightsquigarrow$ root
  - proceed (as usual in AVL) from $x \rightsquigarrow$ root updating Balance Factors and rebalancing – if a rota° is done, then update sizes!

$\Rightarrow$ takes $O(\log n)$

Average / Expected Complexity

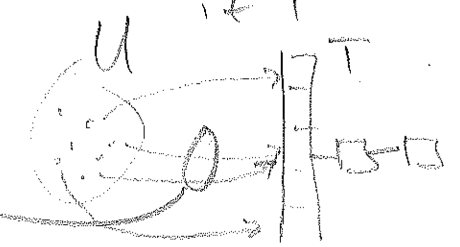
$|U| = u$

e.g. implementing a dictionary with keys $K \subseteq U$, $U = \{0, 1 \dots u-1\}$

1 – u is small, use a direct access table  $= |K|$

2 – u is very large, use a hash table
- let $k \in u$ and T (: hash table); $|T| = m$ (i.e. $[0, \dots m-1]$)
- hash function $h : U \to T$

$$e \mapsto h(e) = i, \quad i \in \{0 \dots m-1\}$$
$$i \in T$$

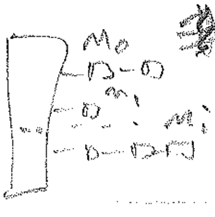
Hashing w/ Chaining $\rightarrow$ 

collision
$\Rightarrow$ linked list
(LIFO)

WC is $O(n)$ (if $h()$ leads to many collisions)

Simple Uniform Hashing Assumption (SUHA)

Any key k is equally likely to hash into any of m slots of T.



⚠ Assumption relies on $h()$ and dist. of $K \subseteq U$.

$$\Rightarrow P(h(k) = i) = \frac{1}{m} \quad (i \in \{0 \dots m-1\})$$

proof $E(n_0 + \dots n_{m-1}) = \sum_{i=0}^{m-1} (E(n_i))$

By SUHA, $\forall i, j$, $E(n_i) = E(n_j)$

$\Rightarrow m E(n_i) = n \Rightarrow E(n_i) = \frac{n}{m} = \alpha$ load factor

Addi° assumption: $m \in \Theta(n) \Rightarrow \alpha \in \Theta(1)$

$\rightarrow$ Worst Case Cost of $RQS(S)$ is $\Theta(m^2)$

AHO, HOPCROFT, ULLMAN

Analysis of RQS

- 2 keys compared iff 1 is a pivot
- 2 keys compared at most once.
- 2 keys split apart by pivot are never compared.

Fix $|S| = m$. cost of RQS $=$ # key comps.

WC : $C = \Theta(m^2)$

AC : $E(C) = ?$

- S has m keys : $z_1 < z_2 < \dots < z_i < \dots z_j < \dots z_m$
- $c_{ij} = \begin{cases} 1 & \text{if } z_i \text{ comp to } z_j \\ 0 & \text{if } \neg \text{ comp to } z_j \end{cases}$

$$\Rightarrow E(c) = E\left[\sum_{i<j} c_{ij}\right] = \sum_{i<j} \underline{E(c_{ij})}$$

$E(c_{ij}) = 1 \times \Pr[c_{ij} = 1] + 0 \times \Pr\dots$
$= \Pr[c_{ij} = 1]$

Consider $Z_{ij} \subseteq Z$ (sorted S), $|Z_{ij}| = j-i+1$

- initially $Z_{ij} \subseteq Z = S$
- RQS keeps selecting pivots ....
  - $\rightarrow$ as long as they aren't in Z_{ij}, Z_{ij} remains entirely cont' in a subseq
  - $\rightarrow$ at some pt, $RQS(S)$ must select a pivot within Z_{ij}
    - case 1: $z_i < p < z_j \Rightarrow z_i$ is never comp to z_j
    - case 2: $p = (z_i \vee z_j) \Rightarrow$ prob of that is $\frac{2}{j-i+1}$

$$\Rightarrow E(c) = \sum_{i=1}^{j} \sum_{j=2}^{m} \frac{2}{j-i+1} = \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} \frac{2}{j-i+1}$$

$$= \sum_{i=1}^{m-1} \sum_{k=1}^{m-i} \frac{2}{k+1} < 2 \sum_{i=1}^{m} \boxed{\sum_{k=1}^{m} \frac{1}{k}}$$

$2 \times m \times H_m$

$$\Rightarrow E(c) \leq 2mH_m \wedge H_m \in O(\ln m)$$

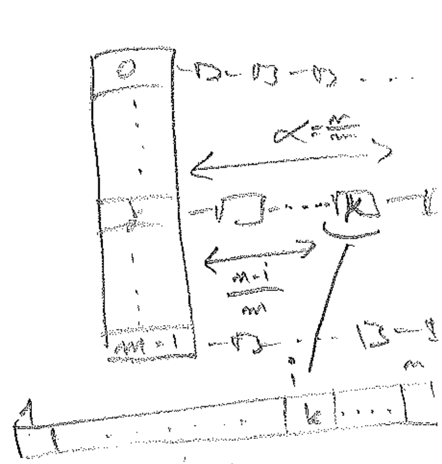
$$E(C) \in O(m \log m)$$

Hashing cont'd

SUHA: $\forall\, 0 \le j \le m-1,\ Pr[h(k) = j] = \frac{1}{m}$

① $k \notin T: E \approx \alpha$

② $k \in T: E \approx \frac{\alpha}{2}$



↳ Assume that $Pr[k = k_i] = \frac{1}{n}$

"k is equally likely to be any of the n key inserted in T."

$\Rightarrow$ Expected cost: $E = \frac{1}{n} \sum_{i=1}^{n} \left[\frac{n-i}{m} + 1\right]$

$= \frac{1}{n}\left(\frac{1}{m}\sum_{i=1}^{n} [n-i] + n\right)$

$= \frac{1}{n} \cdot \frac{1}{m} \cdot \frac{n(n-1)}{2} + 1$

$= \frac{\alpha}{2} - \frac{1}{2m} + 1$ → correcting factor intrin.

$\Rightarrow$ if k was ith key to be inserted, expected # comps of search(k) is $\left[\frac{n-i}{m} + 1\right]$.

distribute k in m slots: $\frac{k}{m}$ and half: $\frac{1}{2m}$

e.g. $\begin{cases} n = 2000 \\ m-1 \text{ is prime} \\ h(k) = k \bmod m \\ E(\text{search}(k)) \le 3 \end{cases} \Rightarrow \alpha \le 3 \Rightarrow m \ge \frac{2000}{3} \Rightarrow m \ge 666 \Rightarrow m = 701$

→ Probabilistic algorithm

▷ Assumes input is random → input follows some 'nice' distrib.

e.g. hashing → SUHA

≠

→ Randomized algorithm

▷ Uses nondeterminism

e.g. ▷ Randomized Quicksort (recursive guy)

input: set of n distinct keys

output: sorted keys

RQS(S). if $S = \emptyset$, then return

. if $|S| = 1$, then output key and return.

. if $|S| \ge 2$, then . select pivot unif. at random.

. by comparing p with every other key in S, split S into: $S_< = \{s \in S \mid s < \text{pivot}\}$

$S_> = \quad "\quad > \quad"$

. $RQS(S_<)$, output p, $RQS(S_>)$

Disjoint Sets

25 years for analysis!
→ 64 - 73 - 79 - 89

- n distinct elements
- initially each is in its own set: $S_1 = \{1\}, S_2 = \{2\} \dots S_n = \{n\}$
- each set has a representative
  - S_x = set rep. by x
- operations:
  - UNION(S_x, S_y): replaces sets S_x and S_y by $S = S_x \cup S_y$
    : returns ptr to set rep of S.
  - FIND(z): finds S, s.t. $z \in S$
    : returns ptr to set rep of S

	S_1	S_2	S_3	S_4	S_5
e.g.	$\{1\}$	$\{2\}$	$\{3\}$	$\{4\}$	$\{5\}$
$U(S_3,S_4)$			$\{3,4\}$	X	
$F(4)=S_3$					
$U(S_1,S_5)$	$\{1,5\}$				X
$U(S_1,S_3)$	$\{1,5,3,4\}$		X		
$F4 = S_1$					

→ At most $n-1$ UNIONs possible
→ σ = seq of $n-1$ UNIONs mixed w/ $m \geq n$ finds
→ how to minimize the cost of executing σ seqs?
→ impl: linked list whose head is the rep of the set

→ cost of UNION: $O(1)$ } $\times n$
→ cost of FIND: $O(n)$ } $\times m$
$\Rightarrow cost(\sigma) = O([m+1]n)$

DT for $m \downarrow\downarrow$

- augmented linked list

→ cost ——— : $O(n)$ } $\times n$
→ cost ——— : $O(1)$ } $\times m$
⇒ ——— $= O(m + n^2)$

- w/ weighted UNION rule — ANALYSIS in TUTORIAL
$O(m + n\log n)$

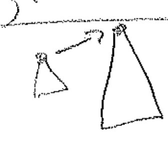
- forest DT

DT for $m \uparrow\uparrow$

→ cost 1 ——— : $O(1)$
→ cost 1 ——— : $O(1 + \text{len of FIND path})$

how to reduce?

① → **Weighted UNION** [by size] CLRS: Rank



Lemma: With WU rule (by size) any tree T of height h created during the exec of seq σ has at least 2^h nodes, ie. $|T| \geq 2^h$

proof (induc° on height of T: h)

- Basis: When a tree of height h = 0 is created it has $2^0 = 1$ node.
- IS: Assume the lemma holds for h

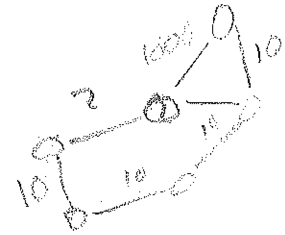
Consider the moment a tree T of height h+1 is created

By IH: $|A| \geq 2^h$

By WU: $|B| \geq |A| \geq 2^h$

$\Rightarrow |T| = |A| + |B|$

$|T| \geq 2^h + 2^h \geq 2^{h+1}$



→ Thus, by lemma: $2^h \leq |T| \leq n \Rightarrow h \leq \log n$

With WU, cost(σ) = $O(n + m \log n)$

② Path compression rule

PC: at each FIND, chg ptrs of all nodes to pt to root)

⟹ With WU & PC, cost(σ) = $O(m \log^* n)$

* ↳ $2 \uparrow\uparrow n$: $\begin{cases} 2^{*0} = 1 \\ 2^{*(n+1)} = 2^{2^{*n}} \end{cases}$

↳ $\lg^* n = \min\{k \mid 2^{*k} \geq n\}$

$2\uparrow\uparrow 0 = 1$
$2\uparrow\uparrow 1 = 2$
$2\uparrow\uparrow 2 = 4$
$2\uparrow\uparrow 3 = 16 = 2^{2^2}$
$2\uparrow\uparrow 4 = 65536 = 2^{2^{2^2}}$
$2\uparrow\uparrow 5 = 2^{65536} \approx 10^{20000}$
$2\uparrow\uparrow 6$: homework

1973: proof of $O(m \log^* n)$ (cf paper)

1975: tarjan: proof of $O(m\, \alpha^{-1}(m, n))$ where α is the ackermann func°

1979: tarjan: ——— Ω(———) under same assump°

→ Double recursion in ackermann func impl. in Machine Switches ~~[illegible]~~ 1911 by Sam.

1989: same w/out assump°

Any alg that solves Disjoint Set Union Find pb is $\Theta(m\, \alpha^{-1}(m, n))$

Minimum cost spanning tree

set of edges s.t. cost is minimized ∧ all cities are connected

$2^3 = 2 \times 2 \times 2$, $2 \uparrow\uparrow 3 = 2^{2^2}$, $2 \uparrow\uparrow\uparrow 3 = 2 \uparrow\uparrow 2 \uparrow\uparrow 2$

$= 2 \uparrow\uparrow (2^{2^2})$

$= 2^{2^{\cdot^{\cdot^{2}}}}$ 16 times

TUTORIAL

no unions: 1 elm / set

1 ——— in worst case, set has at least 2 nodes (by WU)

2 ——— 4

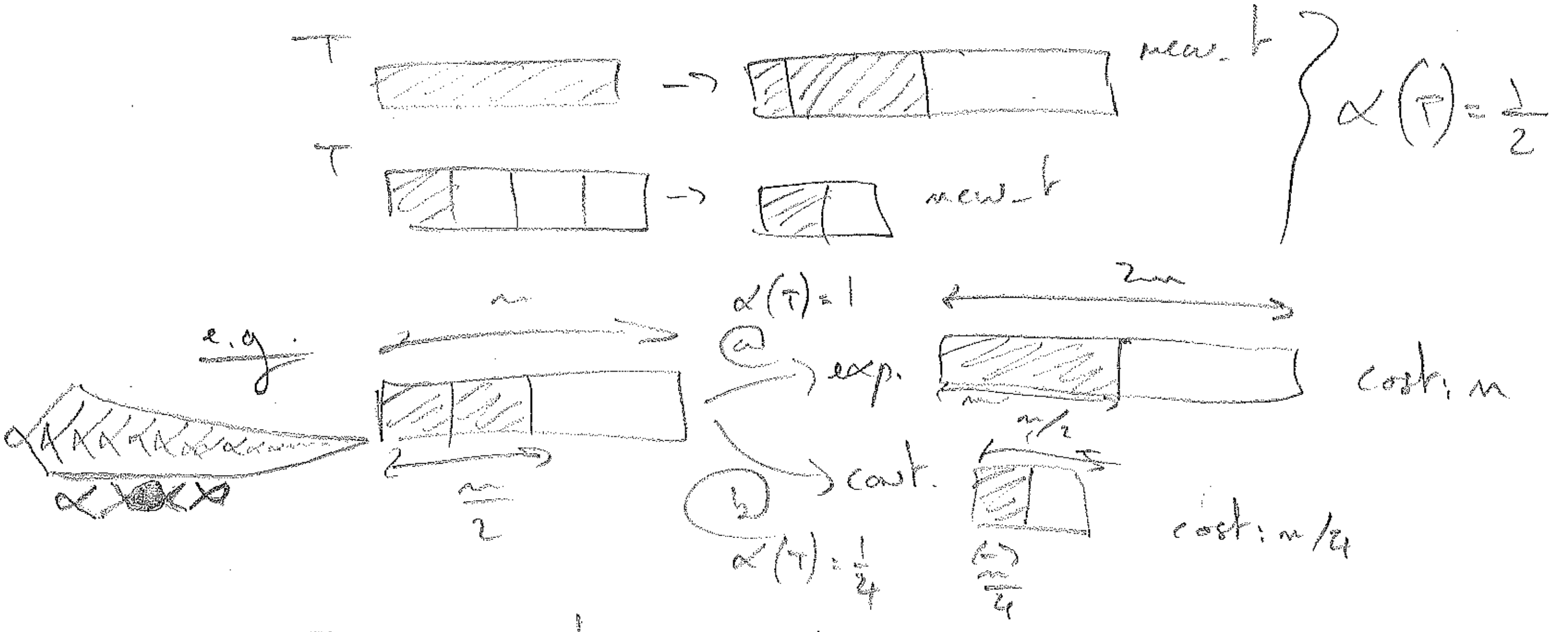
$\Rightarrow 2^k = n \Rightarrow k = \log n$

Kruskal's alg.

Dynamic Table

(a) Expansion: if $\alpha(T) = 1 \wedge$ Insert occurs, size(new_t) = 2 size(T)

(b) Contraction: if $\alpha(T) = \frac{1}{4} \wedge$ Delete —— —— $\frac{1}{2}$ size(T)



ex each insert is charged 3\$

: 1\$ actual cost

: 2\$ credit

each delete ———— 2\$

: 1\$ ——

: 1\$ ——

- If (a) occurs, then at least $\frac{n}{2}$ inserts occurred
these generated a total of $\left(\frac{n}{2}\right) \times 2\$ = n\$$ of credit

→ covers cost of expn.

- I (b) ——— $\frac{n}{4}$ deletes occurred

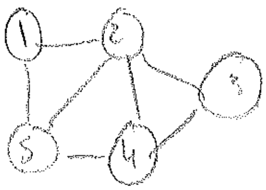
———

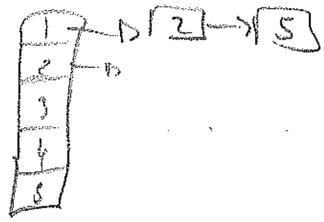
$\left(\frac{n}{4}\right) \times 1\$ = \frac{n}{4}\$$

→ ——— cont.

Graphs $G=(V,E)$, $|V|=n$, $|E|=m$ CSC263

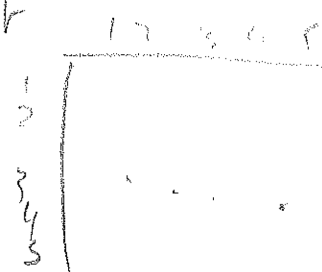
→ adjacency list





$O(n+m)$

adj mat



$O(n^2)$

BFS

start $s \in V$, explore all neighbours

- colour(v) = white → v is undiscovered
  - grey → v was disc. but not explored
  - black → v was disc and expl.
- length of disc. path $s \to v$ = $d[v]$ $s \to u_1 \to \dots \to u \to v$

  $d[v] = d[u]+1$

$P[v]=u$: "u disc. v"

→ BFS tree is graph induced by all edges (u,v) s.t. $P[v]=u$

Q

(to be expl.) OUT ← | u | | | | | v | ← IN (just disc.)

disc ∧ ¬expl.

BFS alg

initialise
```
colour[s] ← grey, d[s] ← 0, P[s] ← NIL
∀ v ∈ V − {s}, colour[v] ← white, d[v] ← ∞, P[v] ← NIL
Q ← EMPTY, ENQ(Q, s)
```

code
```
while Q ≠ ∅
{ u ← DEQ(Q)
  for each (u,v) ∈ E do
  { if colour(v) = white do
     { colour(v) ← grey
       P[v] ← u
       d[v] ← d[u] + 1
       ENQ(Q,v) }
  }
  colour[u] ← BLACK
}
```

WC-complexity: $O(n+m)$

let $\delta(s,v) \triangleq$ length of shortest path $s \to v$ in G

$\Rightarrow \delta(s,v) \le d[v]$

CSC 263

WC-complexity of BFS (graph-disc.) is $O(n+m)$

$u \leadsto v$
$u \not\leadsto v$
$u \leadsto v$

<u>L0</u> $\delta(s,v) \leq d[v]$

<u>L1</u> u enters Q before v during BFS(s) $\Rightarrow d[u] \leq d[v]$

<u>proof</u> Spose ~~by~~ FOR contra that L1 is FALSE

Let v be the 1st node that enters Q s.t. $d[u] > d[v]$ for some u that before v entered Q.

$u \neq s$ bcs $d[s]=0$, $v \neq s$ bcs s is the 1st.

u and v entered Q during exploration of some nodes u' and v' resp.

$d[u] = d[u']+1$ $\quad d[v] = d[v']+1$ $\quad \because d[v] \neq d[u] \Rightarrow u' \neq v'$

Since u entered Q before v, u' was expl. before v', u' entered Q before v' entered Q. By def, $d[u'] \leq d[v']$

$d[u']+1 \leq d[v']+1$

$d[u] \leq d[v]$ → contradiction

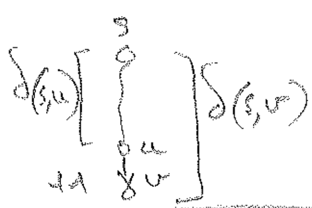
<u>theorem</u> After BFS(s), $\forall v \in V$, $d[v] \stackrel{\delta}{=} \delta(s,v)$

<u>proof</u> Spose for contra, $\exists x \in V$, $d[x] \neq \delta(s,x)$

Let v be closest node from s s.t. $d[v] \neq \delta(s,v)$.

Clearly, $d[v] > \delta(s,v)$ (1)

Shortest



Consider a shortest path from $s \leadsto v$

$\delta(s,v) = \delta(s,u)+1$ (2)

$\boxed{d[v] > d[u]+1}$ *

Since u is closer than v to s, it must be that $\delta(s,u) = d[u]$ (3)

$d[v] > \delta(s,v) = \delta(s,u)+1 = d[u]+1$

→ Consider the colour of v just before u is expl.

(a) V is white: u expl. $v \Rightarrow d[v] = d[u]+1$ — contra to *

(b) V is black: v was expl. bef. u, v ent. Q bef. u

by L1 $\Rightarrow d[v] \leq d[u]$ ——— $\cancel{\;}$

(c) V is grey/gray, some w disc. v bef. u is expl.
$\Rightarrow$ w expl. bef u expl.
$\Rightarrow$ w enters Q bef u enters Q
$\Rightarrow$ $d[w] \le d[u]$, $d[w]+1 \le d[u]+1$ $\Rightarrow$ $d[v] \le d[u]+1$

— contra ✱

Depth First Search

Same as in BFS: colours, $P[v]=u$ iff "u disc. v"

DFS: last

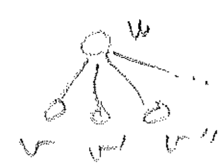
Different from BFS: . global var for time (counter)
$d[u]$: time when u was disc.
$f[u]$: —— u's expl. was compl.

alg . DFS (G)

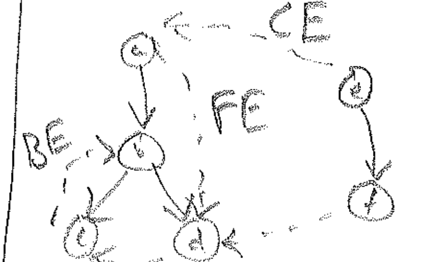
For each $v \in V$
  colour$[v] \leftarrow w$; $d[v] \leftarrow \infty$; $f[v] \leftarrow \infty$; $P[v] \leftarrow$ NIL
time $\leftarrow 0$
For each $v \in V$
  if colour $[v] = w$, then DFS-explore (G, v).

. DFS-explore (G, u)

colour $[u] \leftarrow$ grey
time $\leftarrow$ time $+1$, $d[u] \leftarrow$ time



For each edge $(u, v) \in E$
  if colour $[v]$ = white then /* u disc. v */
    $P[v] \leftarrow u$
    DFS-explore (G, v)
colour $[u] \leftarrow$ black
time $\leftarrow$ time $+1$; $f[u] \leftarrow$ time



FE - forward edges
BE - backward ——
CE - cross ——

$\rightarrow$ DFS forest

a $\rightsquigarrow$ d $\forall (a, d_i)$, a :: ancestor, d :: descendant   CSC 263

[illegible]

$d[a] < d[d] < f[d] < f[a]$

Claim 1   v is a DESC. of u in DFS forest iff
$d[u] < d[v] < f[v] < f[u]$

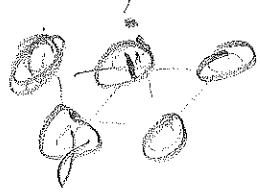
Claim 2   For any 2 nodes, u and v, we cannot have
$d[u] < d[v] < f[u] < f[v]$ → overlapping disr. intervals are imp.

Claim 3   if $(u, v) \in E$, then $d[v] < f[u]$ ~

Applica° of BFS: use to find # comp. (and where)

No cycles $\Leftrightarrow$ $|E| = |V| - \#$ comp

GBM → 5

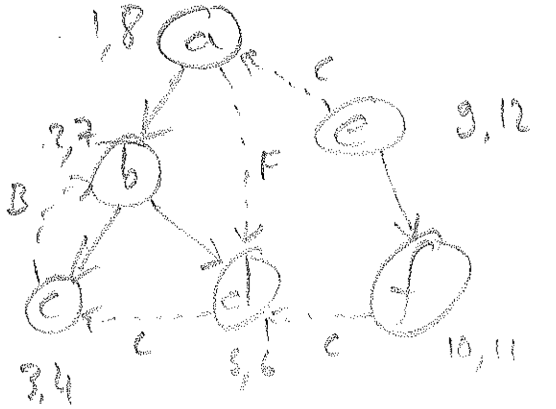


5D hypercube

~~$s_i, s_j = 0$~~

$s_i s_j = 0$ $2^4 - 1$

DFS cont'd



C1: u is ancestor of v in the DFS of G iff $d[u] < d[v] < f[v] < f[u]$

C2: for any u and v, we cannot have $d[u] < d[v] < f[u] < f[v]$

C3: if $(u,v) \in E$ [that is, $u \to v$ in G] then $d[v] < f[u]$

A DFS of a directed graph G classifies its edges as follows:

$(u,v) \in E$ is a:

(1) tree edge iff u is parent of v in DFS of G

non tree edges
- (2) forward edge iff u is ancestor of v ———
- (3) back ——— iff u is descendant of v ———
- (4) cross ——— otherwise

→ Claim: $(u,v) \in E$ is of type

- (1) or (2) iff $d[u] < d[v] < f[v] < f[u]$
- (3) iff $d[v] < d[u] < f[u] < f[v]$
- (4) iff $d[v] < f[v] < d[u] < f[u]$

Application of DFS

→ White Path Theorem (CLRS 22.9)

$\forall G$, $\forall$ DFS of G, v becomes a descendant of u iff at the same time $d[u]$ (the DFS disc. u) there is a path from u to v in G that consists entirely of white nodes.

proof ① ⇒

Spose v is a descendant of u in the DFS at $d[u]$: not yet discovered so they are all white.

② ⇐

Spose at the time $d[u]$, $u \rightsquigarrow w \to z \rightsquigarrow v$

Spose for contra, that ∃ a node in the white path that is not a descendant of u. Let z be the closest such node to u in that path.

Then $w = u$ or w is descendant of u!

- $d[u] < d[z]$ # z is white at time $d[u]$
- $d[z] < f[w]$ # b/c ∃.
- $f[w] \le f[u]$ # because $w = u$ or w descen of u.

⇒ $d[u] < d[z] < f[z] < f[u]$

⇒ z is disc during expl. interval $(d[u], f[u])$

⇒ z is a descendant of u

⇒ contra!

⇒ ∀ node ∈ white path, node is desc. of u.

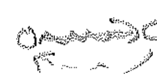
Theorem (CLRS 22.11)

∀ directed G, ∀ DFS of G:

G has a cycle ⇔ DFS of G has a backedge

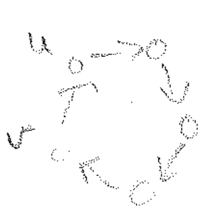
proof ① ⇐

Spose DFS of G finds a back edge (v,u)

then , so G has a cycle

② ⇒

Spose G has a cycle C



Let u be the 1st node in C DFS discovers

Let v be the guy before u in C.

By WPT, v is a desc. of u in the DFS

⇒ backedge by def

Problem Complexity

Let P be a pb. e.g. P: "Sorting n #"

① → Alg complexity: given specific alg A, to solve P (e.g. Heapsort...) what is the cost of solving P using A.

② → Pb complexity: what is the cost of solving P.

Intuitively: By the best possible alg.

→ Decision Tree model:

This can be used to model comparison-based algs. that work by doing comparisons only. ⟹ height of tree is WC.

For each permutaᵒ Π of n; any decision tree for sorting n # has at least 1 leaf corresponding to the inverse permutaᵒ Π^{-1} that 'sorts' input Π. ⟹ the tree has at least n! leaves.

Theorem: Every comparison-based sorting alg A for sorting n # requires at least $\Omega(n \log n)$ comparisons in the WC.

proof Let A be any comp-based alg for sorting n #; T_A be dec-tree of A.

Let $h = \text{height}(T_A)$

note in WC, A takes h comps!

claim: h is $\Omega(n \log n)$

proof of claim:

ⓐ T_A has at least n! leaves (one for each permutaᵒ of n)

ⓑ T_A is a binary tree of height h, so it has at most 2^h leaves ⟹ $2^h \geq n!$ ⟹ $h \geq \lg(n!)$ which is $\Theta(n \lg n)$

→ Adversary Approach

e.g. P = "Find both max & min of set S of n distinct elements"

Naive alg. scan S twice

· n-1 comps

then · n-2 comps

} total: 2n-3 comps.

Better alg

Divide S into $\frac{n}{2}$ pairs

- → Find max, min of each pair. $n/2$
- → Scan $\frac{n}{2}$ maxes to find max: $n/2 - 1$
- → Scan $\frac{n}{2}$ mins to find min: $n/2 - 1$

$\Big\}$ tot.: $\frac{3}{2}n - 2$ comps.

Theorem Any comp-based alg for min-max pb takes at least $\frac{3}{2}n - 2$ comps in WC.

proof of pic.–

Starting from initial state $[n, 0, 0, 0]$

ⓐ The alg must create $n-2$ "M".
To create each "M", it needs to do 1 comp in *
⇒ alg needs to do at least $n-2$ comps of type *
to create $n-2$ "M"s

ⓑ The alg must also create
- $n-2$ → W or L that becomes M
- 1 → W that remains W until the end
- 1 → L ——— L

⇒ alg must create at least n "L" or "W"
⇒ alg needs at least $\frac{n}{2}$ comps to create them.

Minimum Spanning Tree

A tree is a connected undirected graph with no cycles.

tree = undirected acyclic connected graph

spanning tree of G (undirected, connected) = tree $T = (V, E')$, s.t. $E' \subseteq E$

spanning forest = pieces of spanning tree: $F = \bigcup T_i$ ($T_i = (V_i, E_i)$) s.t. $\bigcup_i V_i = V$, $\bigcup E_i \subseteq E$, $V_i \cap V_j = \Phi$

A Minimum Spanning Tree of G is s.t. $\sum w(T)$ is minimized

Num of spanning trees in a clique size n is n^{n-2} ⇒ brute force approach is exponential time

---

Kruskal's MST alg

sort edges alpha + weight and build forest of trees

implementation: array

Given a connected undirected weighted graph $G = (V, E)$

code:

```
Build MinHeap(E)                # linear time
Forest ← { {1}, .... {n} }      # makeSet(V)
MST_edges ← Φ
While |(MST_edges)| < n-1, do
  | (u,v) ← Extract_min (E)
  | Tu ← Find(u) ; Tv ← Find(v)
  | if Tu ≠ Tv, then
  |    | Union (Tu, Tv)
  |    | MST_edges ← MST_edges ∪ {(u,v)}
End
```

loop invar: MST_edges are contained in some MST of G.

WC - Kruskal: $O(m) + O(m) + \underbrace{O(m \log m)}_{\text{dominating term}} + O(m \log^* n)$

$m \in O(n^2) \Rightarrow O(m \log n)$

Traveling Salesman Problem

G :: completely connected w/ nonnegative edge weights.

tour of G: visit every node exactly once.

TSP tour of G: min-cost tour.

TSP → find a TSP tour of G.

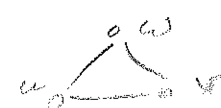
→ brute force: exponential time → $O(n!)$ → bounded by $O(2^n)$

→ polytime alg == $O(n^k)$

TSP is NP-complete.

→ ΔTSP: assume that weights satisfy the triangular inequality

i.e. $\forall u, v, w: c(u,v) \le c(u,w) + c(w,v)$



ΔTSP is NP-complete.

Core Lemma

Let TSP be any opt tour of G

Let MST be any MST of G

Then $\text{cost}(MST) \le \text{cost}(TSP)$

proof. remove 1 edge from

$\forall G, \text{cost}(ST) \le \text{cost}(TSP)$

$\text{cost}(MST) \le \text{cost}(ST)$

Approx alg for ΔTSP with

1. Find MST of G
2. Do a full walk on it → gives a cycle that traverses every edge exactly twice
   ⇒ cycle C

3. Transform C into a tour C^*

use triangular inequality!

Time Complexity

- Worst case asymptotic analysis $O(f(n))$

O: $\exists c$ (factor), $\exists n_0$ (offset), $n > n_0 \Rightarrow f(n) \le g(n)$ — not strict $\le$ 'at some point bounded by'

o: $\forall c$ (for every scaling), $\exists n_0$, $n > n_0 \Rightarrow f(n) < g(n)$ — strict $<$ 'always asymptotically dominates'

$\rightarrow f \in O(n)$ $\rightarrow f \notin o(f)$ $\rightarrow f \in o(g) \Leftrightarrow \lim_{n\to\infty} \frac{f}{g} = 0$

- time complexity class: $TIME(t(n)) = \{L \mid L$ is decidable by some $O(t(n))$ TM$\}$ ($t: \mathbb{N} \to \mathbb{R}^+$)

  note: single tape TM is polynomially equivalent to multitape TM

  $\forall$ multitape TM, $TM \in O(t(n)) \Rightarrow \exists$ single tape $TM' \in O(t^2(n))$ equiv. to TM

- Nondeterministic TM has runtime $f(n)$ = max length of branch of computation
  - $\rightarrow$ constrained to being a decider (arbitrarily?) — accrej?

  $\forall$ nd TM, $TM \in O(t(n)) \Rightarrow \exists$ d TM', $TM' \in O(2^{O(t(n))})$ equiv. to TM

- Sipser: all reasonable models of comp are poly equivalent $\rightarrow$ that are equally powerful

  $f(n) \in o(n \log n) \Rightarrow TIME(f(n)) \subset REG$ $\rightarrow$ $\subseteq$ or $\subsetneq$?

- $P = \bigcup_k TIME(n^k)$
  - $\rightarrow$ invariant for models of comp 'reasonable'
  - $\rightarrow$ most problems in reality aren't very poly big (n^{100} is rare)

$NP = \bigcup_k NTIME(n^k)$ $\rightarrow$ $NTIME(t(n)) = \{L \mid L$ is decided by a $O(t(n))$ nd TM$\}$

$NP = \{L \mid L$ has a polytime verifier$\}$ $\rightarrow$ alg V is a verifier for L iff $L = \{w \mid V$ accepts $\langle w, c \rangle, \exists c \in \Sigma^*\}$

$\rightarrow$ L is polynomially verifiable iff it has a polytime verifier
  - $\rightarrow$ measured only in terms of input w

$CFL \subsetneq P$ $CSL \,?\, NP$

$PATH \in P$, $COPRIME = RELPRIME \in P$